μPnP-Mesh: the Plug-and-Play Mesh Network for the Internet of Things

Nelson Matthys, Fan Yang, Wilfried Daniels,
Sam Michiels, Wouter Joosen, and Danny Hughes
iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium.
Email: {first.last}@cs.kuleuven.be

Thomas Watteyne
Inria Paris-Rocquencourt,
EVA team, France.
Email: thomas.watteyne@inria.fr

***Abstract*—Deploying and customizing networks of Internet of Things (IoT) devices remains extremely challenging. This complexity has two main sources. First, end-users must integrate diverse sensor and actuator peripherals with IoT devices to realize their application. Second, the resulting system must provide reliable mesh networking in harsh network environments at extremely low power. This paper addresses both problems by proposing μPnP-Mesh, which combines the ease of use of μPnP with the industrial performance of SmartMesh IP. μPnP provides a low-power and low-cost method for achieving plug-and-play integration of peripherals with embedded IoT devices; SmartMesh IP provides low-power reliable mesh networking. With a true plug-and-play user experience and a lifetime above 6.5 years on a pair of AA batteries, μPnP-Mesh is a ready-to-use game-changing solution for applications such as home, building and factory automation.**

I. Introduction

The Internet-of-Things (IoT) is composed of resource-constrained embedded devices, equipped with low-power radios and diverse sensors and actuators. These devices typically self-organize and form large networks, capable of monitoring and acting upon phenomena in the physical world. It is commonly accepted that this setting holds great promise for many application scenarios, including smart buildings and smart factories, which are nowadays finding their way into our daily lives.

As technology matures, non-expert IoT users are increasingly in charge of creating and configuring complete applications running on top of embedded IoT devices. This includes the integration of various third-party sensing or actuation peripherals in a “plug-and-play” fashion after the network is already installed. For example, devices deployed in a smart home may initially be configured to monitor temperature. After some time, they may also be equipped with passive RFID sensors and configured to engage in person detection.

Unfortunately, integrating new sensing peripherals on embedded IoT devices is non-trivial and error-prone as this requires knowledge of low-level hardware and software. Each peripheral may also posses specific bandwidth demands. Configuring the networking correctly and guaranteeing that these requirements are met is far from trivial.

A second major hurdle for most contemporary scenarios is that IoT devices are expected to operate over long periods of time on limited energy supplies, and in harsh radio communication environments. Multi-path fading, external interference or other environmental factors [1] make it difficult to achieve reliable wireless communication at low energy costs, a critical requirement for most IoT applications.

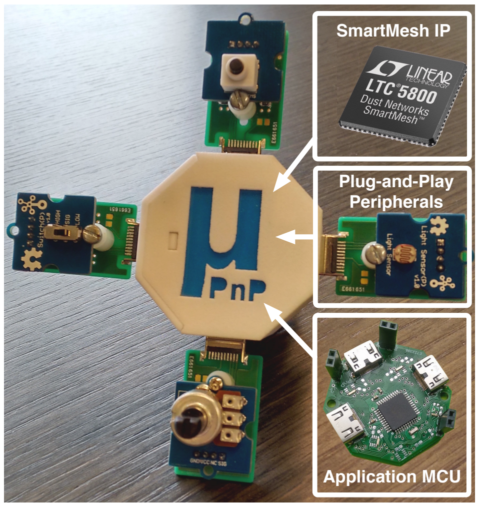


Figure 1. A μPnP-Mesh device with 4 sensor peripherals connected. The μPnP-Mesh device fetches and installs peripheral drivers from the Internet.

Over the last decade, plug-and-play integration of peripherals has gained significant attraction in the context of mainstream systems; solutions such as PCI, USB, or Firewire have been proposed. More recently, lightweight plug-and-play solutions targeting embedded IoT have been proposed by the research community. For example, solutions such as μPnP [2] offer facilities to support auto-identification of peripherals at near-zero energy cost, automatic driver installation, and support for remote peripheral discovery and usage.

Existing solutions for reliable wireless communication are either achieved through contention-based or reservation-based protocols. Resource constraints on embedded IoT devices render contention-based protocols less efficient, while the efficient reservation-based protocols suffer from external interference and multi-path fading, making them unreliable. The Time Synchronized Channel Hopping (TSCH) [3] approach solves this problem by combining a reservation-based Time Division Multiple Access (TDMA) scheme with channel hopping.

This paper proposes the complete design of a new low-power, reliable plug-and-play enabled platform for the IoT. We combine the ease-of-use of μPnP with the industrial performance of SmartMesh IP. μPnP enables a true plug-and-play experience of sensor or actuator peripherals on embedded IoT devices. SmartMesh IP [4] is a proven, market-leading networking solution rooted in TSCH, and which provides ultra low-power and ultra reliable communication for IoT applications. The result is a production-quality and ready-to-use low-power wireless solution.

The remainder of this paper is organized as follows. Section II provides background on low power approaches to mesh networking and plug-and-play sensor integration. Section III describes the μPnP-Mesh system architecture. Section IV presents preliminary evaluation results. Section V concludes and discusses directions for future work.

II. Background

Plug-and-Play IoT Peripheral Solutions. Throughout the years, plug-and-play integration of peripherals has received significant attention in the context of mainstream systems. A number of standardized hardware interconnection technologies – such as the Universal Serial Bus (USB) or IEEE 1394 (FireWire) – have been defined. They provide auto-detection and configuration based on device type identifiers. Despite the increase in flexibility, these conventional approaches either come with large software stacks or consume significant amounts of energy, making them unsuitable for constrained IoT devices that are expected to operate on batteries for years. In contrast, interconnection technologies commonly found on embedded IoT devices – such as ADC, UART, I2C – focus on minimizing CPU, memory and energy consumption, but lack mechanisms to support auto-identification and configuration.

μPnP [2] addresses these issues and provides a mechanism to encapsulate device identifiers by using cheap and low-power passive electrical components. μPnP also comes with a dedicated runtime system, supporting auto-installation and configuration of peripheral drivers, in combination with remote discovery and access through standardized IPv6-based networking. Each time a peripheral is plugged in, the μPnP runtime queries the gateway to find a suitable driver, which is then remotely downloaded. After installation, the peripheral's API is made available over the network.

Evaluation in realistic use-cases shows that μPnP achieves plug-and-play peripheral integration at several orders of magnitude lower energy consumption than embedded USB, and with a flash footprint of only 15 kB [2].

Industrial IoT Network Solutions. Low-power wireless solutions are particularly suited for industrial applications, as they enable "peel-and-stick" deployment, instead of having to install wiring. Industrial applications do require wireless networks to offer the same level of reliability as their wired counterpart, while at the same time offering multiple years of battery lifetime. A new wave of standards and commercial products are appearing, specifically targeted at the Industrial IoT [5].

These standards are rooted in Time Synchronized Channel Hopping (TSCH), a novel communication paradigm particularly well suited for low-power wireless IoT devices. In a TSCH network, all nodes are tightly synchronized, and a schedule orchestrates all communication within the network. The schedule indicates what to do in each timeslot: transmit, listen or sleep. Through time synchronization, nodes only wake up when they effectively communicate, yielding very low power consumption. Successive packets exchanged between two neighbor nodes are done so at different frequencies. The resulting "channel hopping" communication combats external interference and multi-path fading.

SmartMesh IP [4] is a commercial implementation of TSCH, and combines its performance with the ease of use of IPv6. The SmartMesh IP protocol stack is composed of IEEE802.15.4e TSCH at the link layer, and a traditional IPv6-ready IoT "upper stack" (6LoWPAN, UDP). A SmartMesh IP network is composed of a gateway node – the "manager" – and up to 100 "motes". The motes form a redundant low-power wireless multi-hop mesh network.

The manager is responsible for building and maintaining the TSCH schedule of the network. Each mote sends its communication requirements to the manager. The manager computes a TSCH schedule which satisfies the different nodes' requirements, while minimizing the overall energy consumption. It then injects the calculated schedule back into the network. By continuously monitoring the network and adapting the TSCH schedule to topological changes or different communication requirements, a SmartMesh IP network runs autonomously without human intervention.

[6] highlights the performance of SmartMesh IP. In a typical urban deployment in which 100 nodes form a 8-hop deep mesh network, each one publishing a packet every hour with 90 bytes of application payload, the nodes consume between 8.4 μA (nodes furthest from the root) and 22.2 μA (nodes closest to the root). This consumption translates into a battery lifetime of over ten years when using a 2200 mAh battery. In all configurations, a SmartMesh IP network is designed to offer over 99.999% of end-to-end reliability.

III. uPnP-Mesh

μPnP-Mesh combines the ease-of-use of μPnP with the industrial performance of SmartMesh IP. The result is a fully plug-and-play mesh network: a user installs μPnP-Mesh motes is his/her facility, which automatically form an ultra low-power and ultra high reliable IEEE802.15.4 network. He/she then connects sensor and actuator peripherals to the motes, whose corresponding drivers are downloaded automatically. Afterward, each peripheral automatically exposes

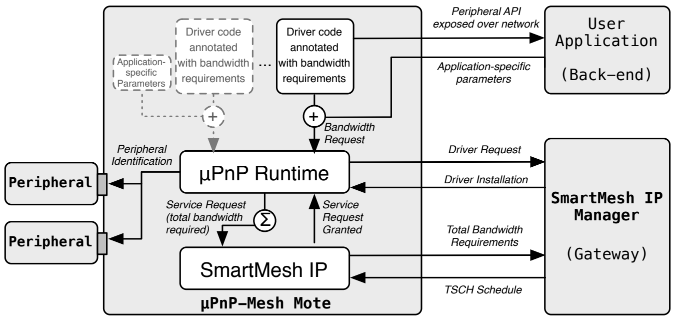

Peripheral API exposed over network
User Application (Back–end)
Application-specific parameters
Driver code annotated with bandwidth requirements
Application-specific Parameters
Driver code annotated with bandwidth requirements
...
Bandwidth Request
Peripheral Identification
µPnP Runtime
Driver Request
Driver Installation
SmartMesh IP Manager (Gateway)
Peripheral
Peripheral
Service Request (total bandwidth required)
Service Request Granted
Total Bandwidth Requirements
SmartMesh IP
TSCH Schedule
µPnP–Mesh Mote

Figure 2. Detailed overview of the μPnP-Mesh solution

the relevant API for data retrieval or actuation. Peripheral drivers are also annotated with information pertaining to their bandwidth requirements. This information is used, in combination with application-specific parameters such as sampling rate, to correctly configure the network.

Section III-A zooms in on the overall system architecture of μPnP-Mesh. Section III-B focuses on how the μPnP-Mesh architecture realizes low-power reliable and QoS-aware networking. Finally, Section III-C describes how drivers inform the allocation of timeslots.

A. System Architecture

As can be seen in Fig. 2, there are three key elements in the μPnP system architecture:

1) **Peripherals** are identified based upon four standard resistors that encode a 32-bit peripheral type identifier. This identifier is read by the μPnP host board connected to the mote. Based upon the type of connected peripheral, it is connected to the appropriate pins on the host micro-controller. Finally the peripheral type identifier is transmitted to the host micro-controller. μPnP currently supports ADC, I2C, SPI and UART peripherals [2].
2) Each **μPnP-Mesh mote** is effectively a wireless hub for locally connected *peripherals*. A mote combines (1) an application micro-controller which runs the μPnP software stack to support peripheral identification, driver installation and remote access to peripheral APIs, and (2) a specialized network chip implementing SmartMesh IP for ultra low power wireless mesh networking.
3) The **SmartMesh IP manager** coordinates network behavior through the allocation of timeslots and application configuration through the deployment of appropriate device drivers to motes with matching peripherals.

B. Low-power, Reliable and QoS-aware Networking

In a SmartMesh IP network, all nodes are tightly synchronized, and time is cut into timeslots. All communication is orchestrated by a communication schedule. The communication schedule indicates to each node what to do in each of the timeslots: transmit, listen or sleep. A timeslot (which is 7.25 ms long) is long enough for the transmitter node to send a data frame to the neighbor it is scheduled to communicate with on that timeslot, and for that neighbor to send back an acknowledgment indicating successful reception. Timeslots are grouped into a "slotframe" (typically 10's to 100's timeslots long) which continuously repeats over time. Between a pair of neighbor nodes, the scheduler can assign multiple timeslots in a slotframe. Assigning more timeslots results in a higher bandwidth (i.e. a larger number of communication opportunities), at the cost of a higher average power consumption.

The goal of the network is to satisfy the communication requirements of the applications running on each node, while maintaining the energy consumption of the nodes as low as possible. At any time, a mote can request a "service". A service is expressed in the number of ms between successive packets. If a mote needs to publish a packet per minute, it requests a service of 60000 ms. If it needs to publish ten packets per second, it requests a service of 100 ms. It is the network that assigns the appropriate number of timeslots to satisfy all the nodes' service requests, including cascading bandwidth requirements in case the requesting mote is multiple hops away from the destination. At any point in time, a SmartMesh IP mote can send a new service request to the network to increase/decrease its bandwidth requirement, which the SmartMesh IP network translates in more/less scheduled timeslots.

μPnP-Mesh uses the "services" mechanism built into SmartMesh IP. A user plugs a sensor into a μPnP-Mesh device, then optionally configures – over the air – how often the device should sample it and publish the measured value. The μPnP middleware contains an "aggregator" module which keeps track of the total bandwidth required by all the applications running on the device. The aggregator issues new service requests as this total bandwidth changes.

C. Calculating the Bandwidth Requirements of Peripheral

The bandwidth requirement of the peripherals is automatically derived from two pieces of information. First, the *data type* used by the remote API of the peripheral driver determines the number of bytes that must be transferred for each operation (e.g. light is encoded in a 16-bit signed integer). Second, the *application specification* running in the resource-rich back-end determines how frequently these API calls are executed (e.g. a temperature sensor deployed in a smart building publishes once per minute). In the current design of μPnP-Mesh, we do not provide support for modeling the bandwidth requirements of stochastic interactions.

The combination of payload and frequency results in a per-driver bandwidth requirement. The μPnP software stack aggregates all bandwidth requirements for host drivers and issues an appropriate service request, as described in Section III-B.

The resulting service request travels back to the manager, which allocates the necessary timeslots to support each peripheral connected to each μPnP-Mesh device. This approach ensures that the network reconfigures to maintain optimal operation, even when heterogeneous sensors – such as for example cameras (high bandwidth) and temperature sensors (low bandwidth) – are connected after deployment.

IV. Evaluation

We have built a production-ready prototype platform of μPnP-Mesh. We describe this platform (Section IV-A), its current draw (Section IV-B), and calculate its battery lifetime in an example scenario (Section IV-C).

A. Production-Ready Prototype Platform

Fig. 1 shows a μPnP-Mesh device prototype. An Atmega1284p micro-controller (10 MHz MCU, 16 kB RAM, 128 kB Flash) implements μPnP on top of Contiki OS [7]. This "μPnP board" is connected to a SmartMesh IP module, which features an LTC5800-IPR chip (ARM Cortex-M3, IEEE802.15.4 compatible radio). Up to four μPnP-enabled peripherals can be connected to a μPnP-Mesh device, as shown in Fig. 1.

B. Instantaneous Current Draw

Table I lists (measured) instantaneous current draw for the main operating modes of the μPnP board and the SmartMesh IP module. The additional current draw of specific peripherals is discussed in Section IV-C. Section IV-C uses those instantaneous current values to build a full energy model and extract the node's lifetime based on its activity.

μPnP board: *peripheral identification* occurs every time a peripheral is plugged into a device. The device remains in this mode until the peripheral has been identified, which takes on average 640 ms. In the *active sensing* mode, the application micro-controller is retrieving data from a sensor. In the *sleep mode*, the application micro-controller is powered down. It can be woken by (i) a wakeup timer, (ii) an interrupt-based sensor or (ii) an incoming message from the SmartMesh IP module.

	peripheral identification	active sensing	sleep mode
μPnP board	8.32 mA	8.30 mA	6.85 μA
	radio on transmitting (+8 dBm)	radio on receiving	radio off (sleep mode)
SmartMesh IP module	9.7 mA	4.5 mA	1.2 μA

Table I
MEASURED INSTANTANEOUS CURRENT FOR THE MAIN OPERATING MODES OF THE μPNP BOARD AND THE SMARTMESH IP MODULE.

SmartMesh IP module: The radio chip on the SmartMesh IP module is in sleep mode the vast majority of the time (with its radio turned off). From time to time, it turns on its radio, either to transmit or to receive. How often this happens depends on the activity of the module, as detailed in Section IV-C.

C. Example Battery Lifetime

We use the energy model for the SmartMesh IP module described in [6], and combine that with the measured values of the μPnP board from Table I to fully characterize the energy consumption (and resulting battery lifetime) of a μPnP-Mesh device.

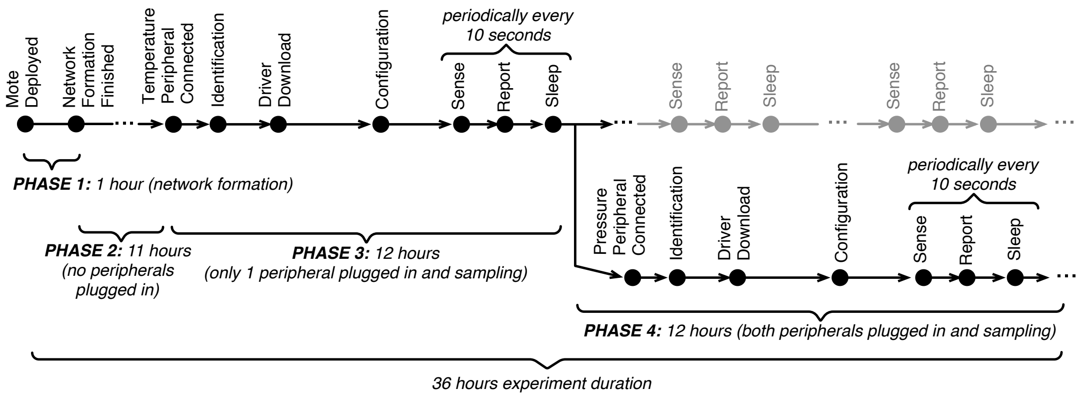
Fig. 3 illustrates a realistic 36-hour usage scenario of a μPnP-Mesh network. By combining the energy model from [6] and the measurements from Table I, we characterize the total charge consumed by a μPnP-Mesh device during that period (Fig. 4).

There are 4 phases in this example use case:

1) For the first hour, 32 μPnP-Mesh devices are deployed and switched on. They are configured to periodically send advertisements (beacons) to allow new devices to join. A device consumes 49.2 μA on average.
2) After one hour, the network has formed. The user enters a command at the manager to turn network advertisement off. The average current draw of a device drops to 33.8 μA.
3) 11 hours after that, a μPnP-ready TMP36 analogue temperature sensor is connected to a device. This device identifies the peripheral and downloads the correct driver from the μPnP back-end, which fits in 3 packets. From that moment on, this device publishes a temperature reading every 10 s.
4) 12 hours later, a μPnP-ready BMP180 I2C pressure sensor is connected to the same μPnP-Mesh device. Again, the peripheral is identified and its driver auto-installed. This device now publishes an additional pressure reading every 10 s.

The total charge consumed by the μPnP-Mesh device the sensors are connected to is shown in Fig. 4. Battery charge is consumed most quickly during the first hour, due to cost of sending advertisements. The small "jumps" at t=12 h and t=24 h reflect the energy spent to identify the peripheral and download its driver. These operations account for 80.08 mC and 129.84 mC for the temperature and pressure sensor, respectively. The pressure sensor consumes more energy during its configuration as its I2C driver is larger and thus requires more packet transmissions.

During those 36 hours, the network has formed, two devices are installed in a plug-and-play fashion and start publishing. This activity consumes less that 0.06% of the available battery charge (a standard 2200 mAh AA battery). The total battery lifetime of the μPnP-Mesh device is over



Mote Deployed
Network Formation Finished
Temperature Peripheral Connected
Identification
Driver Download
Configuration
periodically every 10 seconds
Sense
Report
Sleep
PHASE 1: 1 hour (network formation)
PHASE 2: 11 hours (no peripherals plugged in)
PHASE 3: 12 hours (only 1 peripheral plugged in and sampling)
Pressure Peripheral Connected
PHASE 4: 12 hours (both peripherals plugged in and sampling)
36 hours experiment duration

Figure 3. An example 36-hour usage scenario of a μPnP-Mesh device.

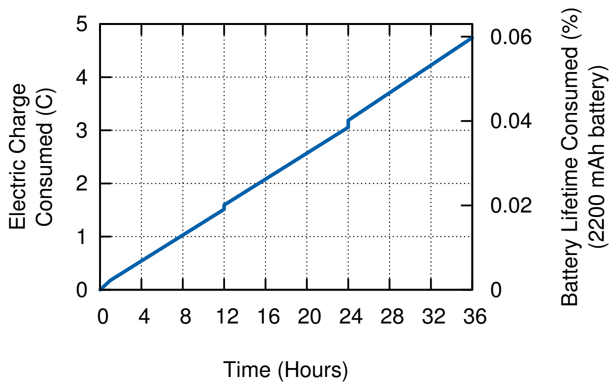


Figure 4. Total battery charge consumed by the scenario illustrated in Fig. 3. We assume two AA batteries (2200 mAh) running at 3.2 V, at 25 C.

6.5 years. μPnP-Mesh achieves true plug-and-play sensor integration while maintaining ultra low-power operation.

V. Conclusion

Applications for the IoT are expected to be deployed in dynamic scenarios and operate reliably for long periods over time. The work presented in this paper provides a real-world state-of-the-art approach to realizing this vision.

μPnP-Mesh makes two key contributions. The first is to combine the benefits of μPnP, i.e. true plug-and-play integration of embedded sensors and actuators, with those of SmartMesh IP, i.e. ultra reliable and ultra low-power mesh networking. The second is to extend the zero-configuration philosophy of μPnP to include Quality of Service (QoS) configuration for Time Synchronized Channel Hopping (TSCH) protocols such as SmartMesh IP.

Our current work focuses on three fronts: (i) real-world large-scale deployments of μPnP-Mesh in realistic application case-studies such as domotics and smart cities, (ii) augmenting the current solution with an algorithm to estimate the bandwidth requirements of stochastically triggered sensor and actuator peripherals, (iii) implementing the μPnP functionality directly on the LTC5800-IPR micro-controller (which also runs the SmartMesh IP protocol stack), so that the application processor can be removed to reduce cost and energy consumption.

Acknowledgments

This research is partially funded by the Research Fund KU Leuven and the iMinds IoT research program.

References

[1] T. Watteyne, A. Mehta, and K. Pister, "Reliability Through Frequency Diversity: Why Channel Hopping Makes Sense," in *Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN).* Tenerife, Canary Islands, Spain: ACM, 29-30 October 2009.

[2] F. Yang, N. Matthys, R. Bachiller, S. Michiels, W. Joosen, and D. Hughes, "μPnP: Plug and Play Peripherals for the Internet of Things," in *European Conference on Computer Systems (EuroSys).* Bordeaux, France: ACM, April 21-24 2015.

[3] K. Pister and L. Doherty, "TSMP: Time Synchronized Mesh Protocol," in *International Symposium on Distributed Sensor Networks (DSN)*, Orlando, FL, USA, 16-18 November 2008.

[4] T. Watteyne, L. Doherty, J. Simon, and K. Pister, "Technical Overview of SmartMesh IP," in *Int'l Workshop on Extending Seamlessly to the Internet of Things (esIoT)*, Taiwan, July 2013.

[5] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6TiSCH: Deterministic IP-enabled Industrial Internet (of Things)," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 36–41, December 2014.

[6] T. Watteyne, J. Weiss, L. Doherty, and J. Simon, "Industrial IEEE802.15.4e Networks: Performance and Trade-offs," in *IEEE Int'l Conference on Communications (ICC)*, 2015.

[7] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors," in *International Conference on Local Computer Networks (LCN).* Tampa, FL, USA: IEEE, 16-18 November 2004.